PRISE EN MAIN DE MAXIMA

Valère Bonnet

4 décembre 2011

Table des matières

1 Avant de commencer

Les dernières versions (et les autres) de *Maxima* et de son interface, *wxMaxima*, sont dans une même archive téléchargeable à
`https://sourceforge.net/project/showfiles.php?group_id=4933&package_id=4960`.
La site officiel de *Maxima* est : `http://maxima.sourceforge.net/`
La site officiel de *wxMaxima* est : `http://wxmaxima.sourceforge.net/`
Maxima fonctionne également en mode console. Pour sortir de *Maxima*, il suffit de lancer « `quit();` » (sans oublier la ponctuation finale).
Lorsqu'on lance *Maxima* ou *wxMaxima*, un message semblable au message suivant apparaît :

```
Maxima 5.16.3 http://maxima.sourceforge.net
Using Lisp CLISP 2.44.1 (2008-02-23)
Distributed under the GNU Public License. See the file COPYING.
Dedicated to the memory of William Schelter.
The function bug_report() provides bug reporting information.
```

Les informations qui suivent sont tirées du mode d'emploi de maxima (874 pages) téléchargeable à :

http://maxima.sourceforge.net/docs/manual/en/maxima.pdf.

2 Calculs élémentaires

2.1 Les quatre opérations

Calculons une somme :

```
(%i1)   2+2;
```

(%o1) $$4$$

Un produit :

```
(%i2)   2*3.4;
```

(%o2) $$6.8$$

Des quotients :

```
(%i3)   -4/6;
```

(%o3) $$-\frac{2}{3}$$

```
(%i4)   -4.0/6;
```

(%o4) $$-0.66666666666667$$

Dans la division euclidienne de 27 par 4 le quotient est 6 et le reste 3 :

```
(%i5)   divide(27,4);
```

(%o5) $$[6,3]$$

On peut aussi diviser des polynômes :

```
(%i6)   divide(3*x^3+4*x^2-5*x+7,2*x^2+x-5);
```

(%o6) $$\left[\frac{6\,x+5}{4},\frac{5\,x+53}{4}\right]$$

Si on a besoin que du reste, on peut utiliser la fonction `mod`.

```
(%i7)   mod(27,4);
```

(%o7) $$3$$

Mais cette fonction n'opère pas avec des arguments polynomiaux.

En utilisant la fonction `display` on peut afficher l'expression à calculer et le résultat.

```
(%i8)   display(sum(k^2,k,0,10));
```

(%o8) $$\sum_{k=0}^{10} k^2 = 385$$

On peut aussi obtenir une somme dépendant d'un paramètre :

```
(%i9)   'sum(k^2,k,1,n)=nusum(k^2,k,1,n);
```

(%o9) $$\sum_{k=1}^{n} k^2 = \frac{n\,(n+1)\,(2\,n+1)}{6}$$

2.2 Factorisations et développements

Décomposons 13 983 816 en produit de facteurs premiers :

```
(%i10)  factor(13983816);
```

(%o10) $$2^3\;3\;7^2\;11\;23\;47$$

Donc : $13\,983\,816 = 2^3 \times 3 \times 7^2 \times 11 \times 23 \times 47$.

Factorisons : $x^6 - 1$.

```
(%i11)  factor(x^6-1);
```

(%o11) $$(x-1)\,(x+1)\left(x^2-x+1\right)\left(x^2+x+1\right)$$

Développons $(x-1)^6$.

```
(%i12)  '(x-1)^6=expand((x-1)^6);
```

(%o12) $$(x-1)^6 = x^6 - 6x^5 + 15x^4 - 20x^3 + 15x^2 - 6x + 1$$

2.3 Valeur absolue ou module

```
(%i13)  abs(-4.5);
```

(%o13) $$4.5$$

```
(%i14)  abs(x^2-x);
```

(%o14) $$\left|x^2-x\right|$$

```
(%i15)  assume(x>1);
```

(%o15) $$[x>1]$$

```
(%i16)  abs(x^2-x);
```

(%o16) $$x^2-x$$

```
(%i17)  forget(x>1);
```

(%o17) $$[x>1]$$

```
(%i18)  abs(x^2-x);
```

(%o18) $$\left|x^2-x\right|$$

2.4 Les racines carrées

On peut également calculer des racines carrées :

```
(%i19)  sqrt(2);
```

```
(%o19)                  √2
(%i20) sqrt(8);
(%o20)                  2^(3/2)
(%i21) sqrt(8.0);
(%o21)                  2.82842712474619
```

Maxima accepte les calculs avec des nombres complexes :

```
(%i22) sqrt(-8);
(%o22)                  2^(3/2) i
(%i23) sqrt(x^2);
(%o23)                  |x|
```

2.5 La fonction partie entière

La partie entière (*floor* en anglais) d'un nombre réel est le plus grand nombre entier relatif inférieur ou égal à ce nombre. La partie entière d'un nombre réel x est notée : $\lfloor x \rfloor$. Par exemple : $\lfloor -2,3 \rfloor = -3$; $\lfloor -5 \rfloor = -5$.

```
(%i24) floor(-2.3);
(%o24)                  -3
(%i25) floor(-5);
(%o25)                  -5
```

On pouvait aussi écrire :

```
(%i26) entier(-2.3);
(%o26)                  -3
(%i27) entier(-5);
(%o27)                  -5
```

la partie entière par excès (*ceiling* en anglais) d'un nombre réel est le plus petit nombre entier relatif supérieur ou égal à ce nombre. La partie entière par excès d'un nombre réel x est notée : $\lceil x \rceil$

```
(%i28) ceiling(-2.3);
(%o28)                  -2
(%i29) ceiling(-5);
(%o29)                  -5
```

Pour de plus amples informations, on pourra consulter :
http://fr.wikipedia.org/wiki/Partie_entière

2.6 Factorielle

Les factorielles sont principalement utilisées en analyse combinatoire. Pour tout entier naturel, n, factorielle n, notée $n!$, est le nombre défini par : $n! = 1 \times 2 \times \cdots \times (n-1) \times n$. Par convention : $0! = 1$.

```
(%i30)    3!;
```

(%o30) 6

```
(%i31)    110!;
```

(%o31) $15882455415227429404253703127 0[119 digits]2192 00000000000000000000000000$

Si on veut afficher tous les chiffres, on peut sélectionner :
Maxima :Basculer en affichage 2d :ascii.

```
(%i32)    set_display('ascii)$
```

(%o32)

```
(%i33)    110!;
```

(%o33) 158824554152274294042537031270907728717244102344735632075817483184445 67\
1629481830309599601315176785204792436726381799902085211486234222668767 57623911\
219200000000000000000000000000

Lorsque n n'est pas un entier, $n!$ est calculer par la fonction gamma d'Euler, voir :
http://fr.wikipedia.org/wiki/Fonction_Gamma_d'Euler.
Il existe aussi la double factorielle : $5!! = 5 \times 3 \times 1$ et $8!! = 8 \times 6 \times 4 \times 2$.

```
(%i34)    10!!;
```

(%o34) 3840

Pour de plus amples informations, on pourra consulter :
http://fr.wikipedia.org/wiki/Factorielle

2.7 Représentation des nombres

Maxima reconnaît les entiers, les rationnels et les réels en général. Ces derniers sont représentés soit par leur valeur exacte, soit par les premiers chiffres de leur développement décimal. *Maxima* représente par défaut les nombres sur 16 chiffres. Pour *Maxima*, 2 est un nombre entier alors que 2.0 est un nombre à virgule.

```
(%i35)    sqrt(43.0)^2-43;
```

(%o35) $-7.105427357601002\,10^{-15}$

```
(%i36)    sqrt(43)^2-43;
```

(%o36) 0

Maxima simplifie automatiquement les fractions.

(%i37) `24/18;`

(%o37) $$\frac{4}{3}$$

La valeur exacte de $\sqrt{126}$ est :

(%i38) `sqrt(126);`

(%o38) $$\sqrt{126}$$

Pour avoir le développement décimal (en virgule flottante) on utilise l'instruction `float`.

(%i39) `float(sqrt(126));`

(%o39) $$11.22497216032182$$

On peut désirer plus que 16 chiffres pour représenter un nombre. On utilise l'instruction `bfloat` (big float).

(%i40) `bfloat(sqrt(126));`

(%o40) $$1.122497216032182b1$$

Le « b1 » final signifie : « $\times 10^1$ ». Le nombre de chiffres significatifs est stocké dans la variable `fpprec` (floating point precision). Par exemple pour obtenir le résultat précédent avec 100 chiffres, il suffit d'écrire :

(%i41) `fpprec:100;`

(%o41) $$100$$

(%i42) `bfloat(sqrt(126));`

(%o42) $$1.1224972160321824156751246196[43digits]68920817083930429685583138 72b1$$

3 Variables, constantes et fonctions

3.1 Variables

Pour déclarer une variable et lui affecter une valeur, on utilise « : ».

(%i43) `a:3;`

(%o43) $$3$$

(%i44) `(a+4)^2;`

(%o44) $$49$$

(%i45) `b:a;`

(%o45) $$3$$

(%i46) `a:a+4;`

(%o46) $$7$$

```
(%i47)  b;
```
(%o47) 3

Pour désaffecter une variable on utilise la commande `kill`.

```
(%i48)  kill(b);
```
(%o48) *done*

```
(%i49)  b;
```
(%o49) b

Certaines variables sont prédéfinies, comme : `fpprec` ou `fpprintprec`.

3.2 Constantes

Les constantes sont généralement précédées du symboles `%`.

```
(%i50)  float(%pi);
```
(%o50) 3.141592653589793

La constante « `%` » est en fait une variable, elle contient le résultat du calcul précédent.

```
(%i51)  %;
```
(%o51) 3.141592653589793

On peut aussi appelé le résultat d'un calcul antérieur.

```
(%i52)  %o49;
```
(%o52) b

Certains constantes sont prédéfinies.

```
(%i53)  float(%e);
```
(%o53) 2.718281828459045

```
(%i54)  %i^2;
```
(%o54) -1

```
(%i55)  float(%phi);
```
(%o55) 1.618033988749895

`%phi` désigne le nombre d'or. Pour de plus amples informations, on pourra consulter :
http://fr.wikipedia.org/wiki/Nombre_d'or

3.3 Fonctions

Pour définir une fonction, on peut utiliser « `:=` » ou « `define` », pour les effacer on utilise « `remfunction` » (remove function).

```
(%i56)  g(x):=2*x+3;
```

(%o56) $$2x+3$$

(%i57)
```
f(x):=(x+1)^2;
```

(%o57) $$f(x) := (x+1)^2$$

(%i58)
```
g(f(x));
```

(%o58) $$2(x+1)^2+3$$

(%i59)
```
remfunction(f);
```

(%o59) $$[f]$$

(%i60)
```
g(f(x));
```

(%o60) $$2f(x)+3$$

On peut aussi définir des fonctions par morceaux. Par exemple, pour introduire la fonction la fonction définie par :

$$f(x) = \begin{cases} -x & \text{si } x < 0 \\ \frac{1}{2}x^2 - x & \text{si } 0 \leqslant x < 2 \\ \frac{x-2}{x-1} & \text{si } 2 \leqslant x \end{cases}$$

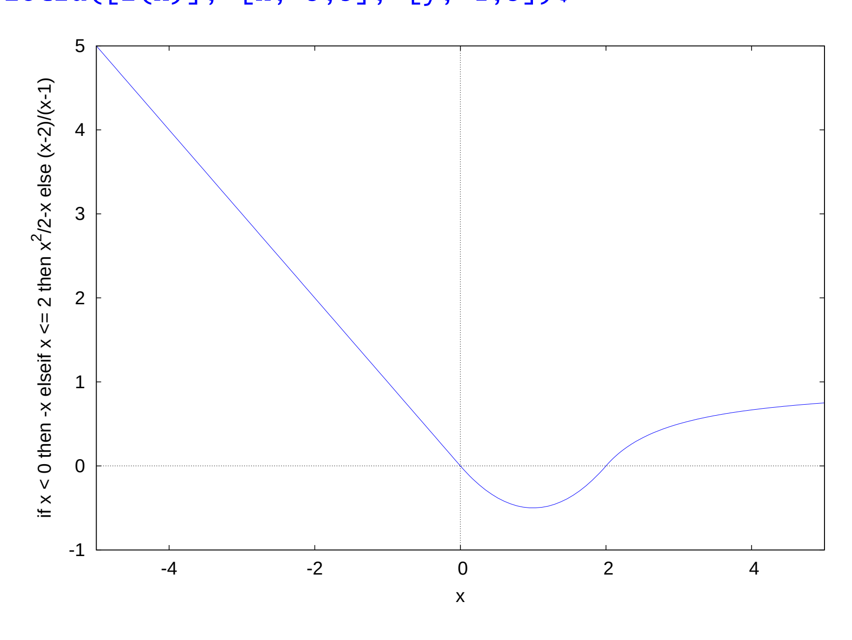
on procède de la façon suivante.

(%i61)
```
f(x):=if x<0 then -x elseif x <=2 then x^2/2-x else (x-2)/(x-1);
```

(%o61) $$f(x) := \ if\ x < 0\ then\ -x\ elseif\ x <= 2\ then\ \frac{x^2}{2} - x\ else\ \frac{x-2}{x-1}$$

Représentons graphiquement cette fonction.

(%i62)
```
wxplot2d([f(x)], [x,-5,5], [y,-1,5])$
```



(%t62)

4 Dérivée d'une fonction

On peut aussi dériver des fonctions.

```
(%i63)  diff(x^3,x);
```

(%o63) $$3x^2$$

```
(%i64)  f(x):=a*x^3+b*x^2+c*x;
```

(%o64) $$f(x) := ax^3 + bx^2 + cx$$

```
(%i65)  'diff(f(x),x)=diff(f(x),x);
```

(%o65) $$\frac{d}{dx}\left(ax^3 + bx^2 + cx\right) = 3ax^2 + 2bx + c$$

```
(%i66)  'diff(f(x),a)=diff(f(x),a);
```

(%o66) $$\frac{d}{da}\left(ax^3 + bx^2 + cx\right) = x^3$$

Soit $f1$ la dérivée de f.

```
(%i67)  define(f1(x),diff(f(x),x));
```

(%o67) $$f1(x) := 3ax^2 + 2bx + c$$

La dérivée de $f1$ est la fonction de $f2$ qui peut aussi être définie par :

```
(%i68)  define(f2(x),diff(f(x),x,2));
```

(%o68) $$f2(x) := 6ax + 2b$$

5 Limite d'une fonction

Quelques constantes :

infinity : ∞

inf : $+\infty$

minf : $-\infty$

ind : indéfini borné, il n'y a pas de limite

und : indéfini non borné, il n'y a pas de limite

Quelques exemples.

```
(%i69)  limit(1/x,x,inf);
```

(%o69) $$0$$

```
(%i70)  limit(sin(x),x,inf);
```

(%o70) $$ind$$

```
(%i71)  limit(1/x,x,0);
```

(%o71) $$und$$

On peut calculer également des limites à droite ou à gauche.

```
(%i72)    limit(1/x,x,0,plus);
```

(%o72) inf

```
(%i73)    limit(1/x,x,0,minus);
```

(%o73) $-inf$

```
(%i74)    limit(abs(x)/x,x,minf);
```

(%o74) -1

```
(%i75)    limit(abs(x)*x,x,infinity);
```

(%o75) $infinity$

6 Calcul intégral

Pour calculer la valeur exacte d'une intégrale, on peut utiliser la fonction `integrate` qui prend quatre arguments : l'expression de la fonction, la variable d'intégration, les bornes inférieure et supérieure. On sait que :

$$\int_1^4 x^2 \, \mathrm{d}\, x = \left[\frac{x^3}{3}\right]_1^4 = \frac{64}{3} - \frac{1}{3} = 21$$

```
(%i76)    integrate(x^2,x,1,4);
```

(%o76) 21

On sait que la primitive de la fonction carré nulle en 1 est la fonction $x \longmapsto \int_1^x t^2 \, \mathrm{d}\, t$.

```
(%i77)    integrate(t^2,t,1,x);
```

(%o77) $\frac{x^3}{3} - \frac{1}{3}$

Malheureusement, *Maxima* ne réussit pas toujours à calculer la valeur exacte de l'intégrale qu'on lui soumet. On aimerait savoir combien vaut : $\int_0^\pi \mathrm{e}^{\sin(x)} \, \mathrm{d}\, x$.

```
(%i78)    integrate(exp(sin(x)),x,0,%pi);
```

(%o78) $\int_0^\pi \% \mathrm{e}^{\sin(x)} \, \mathrm{d}\, x$

Lorsqu'on ne peut obtenir la valeur exacte d'une intégrale, on peut toutefois en entreprendre la détermination d'un valeur approchée. On peut utiliser la `romberg`.

```
(%i79)    romberg(exp(sin(x)),x,0,%pi);
```

(%o79) 6.2087580796135

Pour aller plus loin avec cette fonction

7 Listes

Une liste est une collection d'objets numérotés. Pour déclarer une liste, la collection doit être écrite entre crochets et les éléments doivent être séparés par des virgules. Dans l'exemple suivant, on affecte à une variable « ppp » une liste constituée de trois chaînes de caractères.

```
(%i80)   ppp:["Pim","Pam","Poum"];
```

(%o80) $[Pim, Pam, Poum]$

Pour connaître le premier élément de la liste, il suffit de lancer la commande suivante.

```
(%i81)   ppp[1];
```

(%o81) Pim

On aurait pu également utiliser la commande `first`.

```
(%i82)   first(ppp);
```

(%o82) Pim

Pour connaître le second élément, on lance :

```
(%i83)   second(ppp);
```

(%o83) Pam

Pour connaître le nombre d'éléments de la liste, il suffit de lancer la commande suivante.

```
(%i84)   length(ppp);
```

(%o84) 3

Pour concaténer des listes on utilise : append.

```
(%i85)   append(ppp,["toto"]);
```

(%o85) $[Pim, Pam, Poum, toto]$

On peut également créer des listes de nombres.

```
(%i86)   list1:[7,10,5,21];
```

(%o86) $[7, 10, 5, 21]$

Pour effectuer la somme des termes de cette liste, il suffit de lancer la commande suivante.

```
(%i87)   apply("+",list1);
```

(%o87) 43

Considérons la fonction $f : x \mapsto x^2 - 1$.

```
(%i88)    f(x):=x^2-1;
```

(%o88) $$f(x) := x^2 - 1$$

On peut appliquer f à chaque élément de la liste.

```
(%i89)    map(f,list1);
```

(%o89) $$[48, 99, 24, 440]$$

Ou plus simplement :

```
(%i90)    f(list1);
```

(%o90) $$[48, 99, 24, 440]$$

On peut créer des listes de listes. Créons une liste nomée `xy` de 7 2-listes de nombres entiers naturels, dont le premier est choisi aléatoiremnt de 0 à 8 et le suivant de 0 à 12.

```
(%i91)    xy:makelist([random(9),random(13)],k,1,7);
```

(%o91) $$[[8, 9], [8, 12], [6, 11], [7, 9], [7, 2], [3, 8], [4, 12]]$$

Appliquer une fonction directement ou par `map` ne revient pas forcément au même.

```
(%i92)    liste:[]$for i:1 thru 7 do
(liste:append(liste,[[random(9),random(13)]]));
```

(%o92) $$done$$

```
(%i93)    liste;
```

(%o93) $$[[2, 10], [5, 11], [3, 12], [1, 5], [1, 10], [8, 9], [6, 4]]$$

```
(%i94)    first(liste);
```

(%o94) $$[2, 1]$$

```
(%i95)    x:map(first,liste);
```

(%o95) $$[2, 5, 3, 1, 1, 8, 6]$$

On peut inverser l'ordre d'une liste.

```
(%i96)    reverse(x)
```

(%o96) $$[6, 8, 1, 1, 3, 5, 2]$$

On ranger les éléments d'une liste par ordre croissant ou décroissant.

```
(%i97)    sort(x)
```

(%o97) $$[1, 1, 2, 3, 5, 6, 8]$$

```
(%i98)    reverse(sort(x))
```

(%o98) $$[8, 6, 5, 3, 2, 1, 1]$$

On peut ajouter une constante à une liste.

```
(%i99)   2+[1,2,3];
```

(%o99) $[3,4,5]$

On peut même effectuer le produit scalaire de deux listes de même longueur.

```
(%i100)   [1,2,3].[3,4,5];
```

(%o100) 26

Construisons la liste des carrés des 11 premiers entiers naturels.

```
(%i101)   liste:makelist(i^2,i,0,10)
```

(%o101) $[0,1,4,9,16,25,36,49,64,81,100]$

Construisons la liste des carrés des nombre premiers inférieurs ou égaux à 10.

```
(%i102)   makelist(liste[i+1],i,[2,3,5,7]);
```

(%o102) $[4,9,25,49]$

8 Résolutions d'équations

Les informations diffusées dans ce paragraphe sont en grande partie extraites de :
http://maxima.sourceforge.net/docs/manual/en/maxima_20.html#SEC100
Pour résoudre une équation (déterminer l'ensemble des valeurs exactes des solutions de l'équation), on peut utiliser la commande « `solve` ».
La syntaxe de la commande `solve` est :

`solve`(liste des équations, liste des variables)

```
(%i103)   solve([x+y=5,x-y=3],[x,y]);
```

(%o103) $[[x=4,y=1]]$

```
(%i104)   solve(x^2=2*x+2,x);
```

(%o104) $[x=1-\sqrt{3},x=\sqrt{3}+1]$

```
(%i105)   e1:x^2+y=2;
```

(%o105) $y+x^2=2$

```
(%i106)   e2:x-y=6;
```

(%o106) $x-y=6$

```
(%i107)   solve([e1,e2],[x,y]);
```

(%o107) $[[x=-\frac{\sqrt{33}+1}{2},y=-\frac{\sqrt{3}\,\sqrt{11}+13}{2}],[x=\frac{\sqrt{33}-1}{2},y=\frac{\sqrt{3}\,\sqrt{11}-13}{2}]]$

Dans certains cas le résultat obtenu n'est pas (ou difficilement) exploitable.

(%i108) `solve(x^6=x+1,x);`

(%o108) $$[0 = x^6 - x - 1]$$

(%i109) `solve(x^3=3*x+1,x);`

(%o109) $$\left[x = \left(-\frac{\sqrt{3}\,i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{\frac{\sqrt{3}\,i}{2} - \frac{1}{2}}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}}, x = \left(\frac{\sqrt{3}\,i}{2} - \frac{1}{2}\right)\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{-\frac{\sqrt{3}\,i}{2} - \frac{1}{2}}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}}, x = \left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}} + \frac{1}{\left(\frac{\sqrt{3}\,i}{2} + \frac{1}{2}\right)^{\frac{1}{3}}} \right]$$

On peut alors chercher des valeurs approchées des solutions en utilisant l'une des commandes suivantes :

`allroots` `allroots`(expr) ou `allroots`(eqn) ;

`bfallroots` `bfallroots`(expr) ou `bfallroots`(eqn) ;

`realroots` `realroots`(expr) ou `realroots`(eqn) ou `realroots`(expr, incertitude) ou `realroots`(eqn, incertitude) ;

`find_root` `find_root`(expr, x, a, b, [abserr, relerr]), `find_root`(f, a, b, [abserr, relerr]) ;

`bf_find_root` `bf_find_root`(expr, x, a, b, [abserr, relerr]), `bf_find_root`(f, a, b, [abserr, relerr]) ;

(%i110) `f(x):=x^6-x-1`

(%o110) $$x^6 - x - 1$$

(%i111) `allroots(f(x));`

(%o111) $[x = 1.002364571587165\,\%i + 0.45105515860886, x = 0.45105515860886$
$-1.002364571587165\,\%i, x = 0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.7780895986786, x = 1.134724138401519]$

(%i112) `allroots(x^6=x+1);`

(%o112) $[x = 1.002364571587165\,\%i + 0.45105515860886, x = 0.45105515860886$
$-1.002364571587165\,\%i, x = 0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.7780895986786, x = 1.134724138401519]$

(%i113) `bfallroots(f(x));`

(%o113) $[x = 1.002364571587165b0\,\%i + 4.510551586088557b-1,$
$x = 4.510551586088557b-1 - 1.002364571587165b0\,\%i,$
$x = 7.357559529997765b-1\,\%i - 6.293724284703148b-1,$
$x = -7.357559529997765b-1\,\%i - 6.293724284703148b-1,$
$x = -7.780895986786011b-1, x = 1.134724138401519b0]$

(%i114) `bfallroots(x^6=x+1);`

(%o114) $[x = 1.002364571587165\,\%i + 0.45105515860886, x = 0.45105515860886$
$-1.002364571587165\,\%i, x = 0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.73575595299978\,\%i - 0.62937242847031,$
$x = -0.7780895986786, x = 1.134724138401519]$

(%i115) `fpprec:30`

(%o115) 30

(%i116) `bfallroots(x^6=x+1);`

(%o116) $[x = 1.00236457158716501942954562416b0$
$\%i + 4.51055158608855643526288390702b-1,$
$x = 4.51055158608855643526288390702b-1$
$-1.00236457158716501942954562416b0\,\%i,$
$x = 7.35755952999776458609773027245b-1$
$\%i - 6.29372428470314840888670263125b-1,$
$x = -7.35755952999776458609773027245b-1$
$\%i - 6.29372428470314840888670263125b-1,$
$x = -7.78089598678601097880682309659b-1,$
$x = 1.13472413840151949260544605451b0]$

(%i117) `realroots(x^6=x+1);`

(%o117) $[x = -\frac{26108355}{33554432}, x = \frac{38075023}{33554432}]$

`find_root` et `bf_find_root` donnent une solution réelle d'une équation dans un intervalle fermé donné. On peut par exemple calculer une valeur approche de la solution réelle de l'équation précédente qui est dans l'intervalle [0 ; 3].

(%i118) `find_root(f(x),0,3);`

(%o118) 1.134724138401519

(%i119) `find_root(x^6=x+1,0,3);`

(%o119) 1.134724138401519

(%i120) `bf_find_root(x^6=x+1,0,3);`

(%o120) $1.13472413840151949260544605451b0$

On remarquera que dans une expression du type,
bf_find_root(f, a, b, [abserr, relerr]),

les bornes a et b doivent avoir des images de signes contraires par f.

9 Modules

9.1 Introduction

Divers fichiers de documentation sont présents dans l'arborescence du répertoire d'installation, pour en avoir la liste il suffit de lancer :

```
(%i121) printfile("share.usg");
ABSIMP - (SHARE1;) supplements the built-in simplification rules
for ABS and SIGNUM.
...
WORLD - belongs to PLOT2
```

(%o121) $/usr/share/maxima/5.13.0/share/share.usg$

Pour avoir, par exemple, de plus amples informations sur le module des constantes physiques, il suffit de lancer : « `printfile("physconst.usg");` ». Pour utiliser ce module, il suffit de lancer : « `load("physconst.mac");` ».

9.2 Descriptive

Le module « descriptive [1] » simplifie en les automatisant les calculs des caractéristiques de séries statistiques.

```
(%i122) load(descriptive);
Warning - you are redefining the Maxima function range
```

(%o122) $/usr/local/share/maxima/5.16.3/share/contrib/descriptive/descriptive.mac$

```
(%i123) liste:makelist(2*i,i,10,27);
```

(%o123) $[20, 22, 24, 26, 28, 30, 32, 34, 36, 38, 40, 42, 44, 46, 48, 50, 52, 54]$

moyenne `(%i124) mean(liste);`

(%o124) 37

variance `(%i125) var(liste);`

(%o125) $\frac{323}{3}$

1. Statistique descriptive : La statistique descriptive est la branche des statistiques qui regroupe les nombreuses techniques utilisées pour décrire un ensemble relativement important de données.

écart type (%i126) `std(liste);`

(%o126) $$\frac{\sqrt{323}}{\sqrt{3}}$$

minimum (%i127) `mini(liste);`

(%o127) $$20$$

maximum (%i128) `maxi(liste);`

(%o128) $$54$$

étendue (%i129) `range(liste);`

(%o129) $$34$$

médiane (%i130) `median(liste);`

(%o130) $$37$$

quantile D_1 : (%i131) `quantile(liste,1/10);`

(%o131) $$\frac{117}{5}$$

Nous remarquons que cette fonction est construite avec une autre définition que celle donnée en cours. En effet, l'effectif est de liste est 18 et : $18 \times 0,1 = 1,8$; $18 \times 0,9 = 16,2$; donc il doit y avoir au moins 2 individus réalisant ($liste[i] \leqslant D_1$) et au moins 19 réalisant ($liste[i] \geqslant D_1$). Donc avec la définition du cours, D_1 est le second élément de la liste :

(%i132) `sort(liste)[2]`

(%o132) $$22$$

Cependant, en pratique, la différence entre les deux définitions s'estompe.

9.3 Distrib

Le module

9.4 Numericalio

Comme son nom l'indique, le module « numericalio » gère les entrées et les sorties de données numériques (des nombres) entre *Maxima* et les disques durs. Ce module n'est pas chargé par défaut. Il faut lancer :

```
(%i133) load("numericalio");
(%o133)     /usr/share/maxima/ ··· /share/contrib/numericalio/numericalio.lisp
```

On peut, entre autre, avec ce module lire ou écrire des listes sur un disque dur. Enregistrer les fichiers :
http://mathsaulycee.info/memo/605/list1.txt
http://mathsaulycee.info/memo/605/list2.txt
Dans le répertoire : `U:\votrenom`.
Ouvrir ces fichiers pour en voir le contenu, puis lancer :

```
(%i134) essai1read_list("U/votrenom/list1.txt");
(%o134)     ...
(%i135) essai2read_list("U/votrenom/list2.txt");
(%o135)     ...
(%i136) essai3read_nested_list("U/votrenom/list2.txt");
(%o136)     ...
```

Expliquez les différences.

On peut également utiliser ce module pour écrire des données sur un disque dur en utilisant par exemple `write_data`. Pour sauvegarder la liste `ppp` dans un fichier nommé `toto.txt` il suffit de lancer :

```
(%i137) write_data(ppp,"U/votrenom/toto.txt");
(%o137)     done
```

9.5 Romberg

Le module « romberg » est utiliser pour calculer des valeurs approchées d'intégrales.

10 Suites numériques

10.1 Définitions

Une suite peut être définie soit comme une fonction, soit comme une liste infinie. Considérons par exemple la suite $(u_n)_{n\in\mathbb{N}}$ de terme général : $u_n = 2n - 3$.
On peut l'introduire comme une fonction :

```
(%i138) u(n):=2*n-3;
```

(%o138) $u(n) := 2 * n - 3$

ou comme une liste infinie :

```
(%i139) u[n]:=2*n-3;
```

(%o139) $u_n := 2 * n - 3$

Nous constatons à travers cet exemple que la deuxième option est plus recommandable. Pour connaître u_{12}, il suffit d'entrer :

```
(%i140) u[12];
```

(%o140) 21

Ici le problème était simple, car la suite (u_n) est définie explicitement. Considérons maintenant une suite définie par récurrence, par exemple, la suite (v_n) définie par : $\begin{cases} v_0 = 4 \\ v_n = 2v_{n-1} - 3 \end{cases}$.

Pour définir cette suite, nous allons utiliser une propriété de *Maxima* : la récursivité.

Pour de plus amples informations, on pourra consulter :

http://fr.wikipedia.org/wiki/Récursivité.

Un langage récursif est un langage qui acceptent des processus récursifs.

On peut donc définir (v_n) de la façon suivante :

```
(%i141) v[n]:= if n=0 then 4 else 2*v[n-1]-3;
```

(%o141) $v_n := \text{ } if \ n = 0 \ then \ 4 \ else \ 2\, v_{n-1} - 3$

Pour connaître la valeur de v_{16}, il suffit d'entrer :

```
(%i142) v[16];
```

(%o142) 65539

Lors de cette dernière commande, ce n'est pas simplement v_{16} qui a été calculé ; mais toutes les valeurs de v_0 à v_{16}. Pour contourner ce problème, on aimerait avoir l'expression explicite de la suite (v_n). Pour réussir dans cette démarche on peut parfois utiliser le module « solve_rec ».

```
(%i143) load("solve_rec");
```

(%o143) /usr/local/share/maxima/5.16.3/share/contrib/solve_rec/solve_rec.mac

```
(%i144) solve_rec(a[n]=2*a[n-1]-3,a[n]);
```

(%o144) $a_n = \%k_1\, 2^n - 3\, 2^n + 3$

Cette dernière réponse signifie que le terme général de la suite (u_n) est de la forme :

$$u_n = k_1 \times 2^n - 3 \times 2^n + 3.$$

où k_1 est une constante à déterminer. Mais en affectant à n la valeur 0, il vient : $k_1 = u_0 = 4$. Nous en déduisons l'expression explicite du terme

général de la suite :

$$u_n = 2^n + 3.$$

Il existe cependant une manière plus simple d'aboutir au résultat :

```
(%i145)   solve_rec(a[n]=2*a[n-1]-3,a[n],a[0]=4);
```

(%o145) $$a_n = 2^{n+2} - 3\,2^n + 3$$

Pour simplifier le résultat obtenu et l'affecter à la suite (v_n), on lance :

```
(%i146)   define(v[n],radcan(rhs(%)));
```

(%o146) $$v_n := 2^n + 3$$

Dans cette dernière expression, `rhs(%)` désigne le membre de droite (right hand side) de l'égalité %o122 et la fonction `radcan` simplifie les expressions. Il existe de même la fonction `lhs` (left hand side).

10.2 Suites particulières

Les fonctions suivantes sont définies dans le module « functs ».

`arithmetic`(a, r, n) retourne le n-ième terme de la suite arithmétique de raison r et de premier terme a, c'est-à-dire : $u_{n-1} = u_0 + (n-1)u_0 = a + r(n-1)$.

```
(%i147)   load("functs")
```

(%o147) $/usr/local/share/maxima/5.16.3/share/simplification/functs.mac$

```
(%i148)   arithmetic(1,2,3)
```

(%o148) $$5$$

`geometric`(a, q, n) retourne le n-ième terme de la suite géométrique de raison q et de premier terme a, c'est-à-dire : aq^{n-1}.

```
(%i149)   geometric(1,2,3)
```

(%o149) $$4$$

`arithsum`(a, r, n) retourne la somme des n premiers termes de la suite arithmétique de raison r et de premier terme a, c'est-à-dire :

$$\sum_{k=1}^{n} \left(a + r(k-1)\right) = n\left(a + \frac{n-1}{2}r\right).$$

```
(%i150)   arithsum(1,2,3)
```

(%o150) $$9$$

`geosum`(a, q, n) retourne la somme des n premiers termes de la suite géométrique de raison q et de premier terme a.

```
(%i151)   geosum(1,2,3)
```

(%o151) $$\frac{7}{\frac{a}{b+(n-1)c}}$$

harmonic(a,b,c,n) retourne la valeur de : $\frac{a}{b+(n-1)c}$.

(%i152) `harmonic(1,2,3,4)`

(%o152) $$\frac{1}{11}$$

(%i153) `harmonic(1,1,1,1000)`

(%o153) $$\frac{1}{1000}$$

10.3 Suites mutuellement définies

On peut utiliser ces techniques pour traiter des suites mutuellement définies. Il suffit pour cela de regrouper les deux suites en une seule dont chaque terme est une 2-liste. Considérons par exemple les suites (u_n) et (v_n) définies par : $u_0 = 1$; $v_0 = 63$ et pour tout entier naturel, n :

$$u_{n+1} = \frac{u_n + v_n}{2} \qquad v_{n+1} = \frac{u_n + 3v_n}{4}.$$

Le module solve_rec ne semble pas fonctionner sur les suites de 2-listes. Nous pouvons toutefois écrire :

(%i154) `A:matrix([1/2,1/4],[1/2,3/4]);`

(%o154) $$\begin{bmatrix} \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & \frac{3}{4} \end{bmatrix}$$

(%i155) `a[n]:=if n=0 then [1,63] else a[n-1].A;`

(%o155) $$a_n := \text{if } n = 0 \text{ then } [1,63] \text{ else } a_{n-1}.A$$

(%i156) `a[10];`

(%o156) $$\begin{bmatrix} \frac{11097419}{262144} & \frac{22194869}{524288} \end{bmatrix}$$

(%i157) `bfloat(%);`

(%o157) $$\begin{bmatrix} 4.233329391479492b1 & 4.233335304260254b1 \end{bmatrix}$$

On peut aussi procéder en deux temps :

(%i158) `u[n] := if n=0 then 1 else (u[n-1]+v[n-1])/2;`

(%o158) $$u_n := \ if \ n = 0 \ then \ 1 \ else \ \frac{u_{n-1} + v_{n-1}}{2}$$

(%i159) `v[n] := if n=0 then 63 else (u[n-1]+3v[n-1])/4;`

(%o159) $$v_n := \ if \ n = 0 \ then \ 63 \ else \ \frac{u_{n-1} + 3v_{n-1}}{4}$$

```
(%i160) u[3];
```

(%o160) $$\frac{667}{16}$$

11 Arithmétique

Pour savoir si un nombre est premier :

```
(%i161) primep(547);
```

(%o161) *true*

Nombre premier précédent et nombre premier suivant :

```
(%i162) prev_prime(18);
```

(%o162) $$17$$

```
(%i163) next_prime(18);
```

(%o163) $$19$$

Ensemble des diviseurs naturels d'un nombre :

```
(%i164) divisors(24);
```

(%o164) $$\{1, 2, 3, 4, 6, 8, 12, 24\}$$

`totient`(n) désigne le nombre d'entiers naturels premiers avec n et plus petit que n.

```
(%i165) totient(24);
```

(%o165) $$8$$

Pour décomposer un entier en produit de facteurs premiers :

```
(%i166) factor(720);
```

(%o166) $$2^4 3^2 5$$

```
(%i167) ifactors(720);
```

(%o167) $$[[2, 4], [3, 2], [5, 1]]$$

Dans la division euclidienne de 27 par 4 le quotient est 6 et le reste 3 :

```
(%i168) divide(27,4);
```

(%o168) $$[6, 3]$$

Le PGCD de 24 et 42 est 6 :

```
(%i169) gcd(24,42);
```

(%o169) $$6$$

Pour obtenir le PPCM, on charge le module « functs ».

```
(%i170) load("functs");
```

(%o170) $/usr/local/share/maxima/5.16.3/share/simplification/functs.mac$
(%i171) `lcm(24,42)`

(%o171) 168

Le reste modulo 9 de 56 est 2.

(%i172) `mod(56,9);`

(%o172) 2
(%i173) `mod(56^2009,9);`

(%o173) 5

D'après le théorème de Bézout-Bachet, pour tous entiers non nuls a et b de PGCD δ, il existe un unique couple d'entiers (u, v) tel que :

$$au + bv = \delta \quad \text{avec} \quad |u| \leqslant |b| \quad \text{et} \quad |v| \leqslant |a|$$

(%i174) `gcdex(124,74);`

(%o174) $[3, -5, 2]$

On a en déduit que : PGCD $(124, 74) = 2$ et $3 \times 124 - 5 \times 74 = 2$.

12 Combinatoire

Les factorielles ont été vues § 2.6 page 6. Les calculs de nombres de combinaisons ou d'arrangements sont fournis par le module « functs ». Une combinaison de p éléments parmi n est un sous-ensemble à p éléments d'un ensemble à n éléments. Le nombre noté, $\binom{n}{p}$ ou C_n^p, désigne le nombre de combinaisons à p éléments qu'il est possible de former dans un ensemble à n éléments. On a :

$$\binom{n}{p} = \frac{n!}{p!(n-p)!}.$$

(%i175) `load("functs")`

(%o175) $/usr/local/share/maxima/5.16.3/share/simplification/functs.mac$
(%i176) `combination(5,2)`

(%o176) 10

Un arrangement de p éléments parmi n est une p-liste d'éléments distincts d'un ensemble à n éléments. A_n^p, désigne le nombre d'arrangements à p éléments qu'il est possible de former dans un ensemble à n éléments. On a :

$$A_n^p = \frac{n!}{(n-p)!}.$$

```
(%i177)    permutation(5,2)
(%o177)                                20
```

Index

Index des commandes Maxima

Index des modules Maxima